

# Automated Cryptanalysis of Bloom Filter Encryptions of Health Records

Martin Kroll      Simone Steinmetzer

*University of Duisburg-Essen*

{martin.kroll, simone.steinmetzer}@uni-due.de

October 27, 2014

Privacy-preserving record linkage with Bloom filters has become increasingly popular in medical applications, since Bloom filters allow for probabilistic linkage of sensitive personal data. However, since evidence indicates that Bloom filters lack sufficiently high security where strong security guarantees are required, several suggestions for their improvement have been made in literature. One of those improvements proposes the storage of several identifiers in one single Bloom filter. In this paper we present an automated cryptanalysis of this Bloom filter variant. The three steps of this procedure constitute our main contributions: (1) a new method for the detection of Bloom filter encryptions of bigrams (so-called *atoms*), (2) the use of an optimization algorithm for the assignment of atoms to bigrams, (3) the reconstruction of the original attribute values by linkage against bigram sets obtained from lists of frequent attribute values in the underlying population. To sum up, our attack provides the first convincing attack on Bloom filter encryptions of records built from more than one identifier.

*Keywords:* Bloom Filter, Privacy-Preserving Record Linkage, Anonymity, Hash Function, Cryptographic Attack

## 1 Introduction

Record linkage between databases containing information on individual people is popular in a large number of medical applications, for example the identification of patient deaths [4], the evaluation of disease treatment [10] and the linkage of cancer registries in epidemiology [18]. In many applications data sets are merged using personal identifiers such as forenames, surnames, place and date of birth. Due to privacy concerns this has to be done via privacy-preserving record linkage (PPRL). However, since personal identifiers often contain typing or spelling errors, encrypting the identifier values and linking only those that match exactly does not provide satisfactory results. Therefore, to allow for errors in encrypted personal identifiers, in many European countries encrypted phonetic codes, such as Soundex codes, are commonly used, especially by cancer registries. As the performance of these codes is still non satisfactory, several novel privacy-preserving record linkage methods have

been suggested during the last years. For example Schnell et al. [15] developed a method based on Bloom filters. Bloom-filter-based record linkage has already been used in medical applications in a number of different countries [6, 9, 13, 17].

Another frequently applied privacy-preserving record linkage method uses anonymous linking codes [2]. The basic principle of an anonymous linking code is to standardize all particular identifiers of a record (removal of certain characters and diacritics, use of upper case letters), to concatenate them to a single string and finally to put this single string into a cryptographic hash function. By combining this principle with Bloom filters, Schnell et al. [16] first developed a novel error-tolerant anonymous linking code, called Cryptographic Longterm Key (CLK). Instead of encrypting every single identifier from a record of several identifiers through a Bloom filter, multiple identifiers are stored in one single Bloom filter, called CLK. Tests on several databases showed that CLKs yield good linkage properties, superior to well-known anonymous linking codes [16].

Only recently Randall et al. [13] presented a study on 26 million records of hospital admissions data and showed that privacy-preserving record linkage with Bloom filters built from multiple identifiers is applicable to large real-world databases without loss in linkage quality.

However, only little research on the security of Bloom filters built from more than one identifier has yet been published (see subsection 2.2). In several countries, this lack of research prevents the widespread use of Bloom filter encryptions for real-world medical databases (such as cancer registries) where the anonymity of the individuals has to be guaranteed. For example, in its *Beyond 2011 Programme* the British Office for National Statistics investigated several methods for linking sensitive data sets [12]. The investigators came to the conclusion that none of the "(...) recent innovations, such as bloom filter encryption (...)" can be recommended because they "(...) have not been fully explored from an accreditation perspective". Thus, research showing drawbacks of the recent Bloom filter techniques is important because it guides the direction for future research and might motivate further development of the recent procedures. In this paper, we intend to investigate this issue in detail by giving the first convincing cryptanalysis of Bloom filter encryptions built from more than one identifier.

## 2 Background

In 1970, Burton Howard Bloom [1] introduced a novel approach that permits the efficient testing of set membership through a probabilistic space-efficient data structure. A *Bloom filter* is a bit array of length  $L$ , which at first contains zeros only. Let  $S \subseteq \mathcal{U}$  be a subset of a universe  $\mathcal{U}$ . Then  $S$  can be stored in a Bloom filter  $\mathcal{B} = \mathcal{B}(S) = (b_0, \dots, b_{L-1})$  in the following way: Each element  $s \in S$  is mapped via  $k$  different hash functions  $h_0, \dots, h_{k-1} : S \rightarrow \{0, \dots, L-1\}$  and all the corresponding bit positions  $b_{h_0(s)}, \dots, b_{h_{k-1}(s)}$  are set to one. Once a bit position is set to one this value no longer changes.

Furthermore, to test whether an item  $u \in \mathcal{U}$  from the universe is contained in  $S$ ,  $u$  is hashed through the  $k$  hash functions  $h_0, \dots, h_{k-1}$  as well. Consequently, if all bit positions  $b_{h_0(u)}, \dots, b_{h_{k-1}(u)}$  in the Bloom filter are set to one, then  $u \in$

$S$  holds with high probability. However, false positive values can occur when the ones on positions  $h_0(u), \dots, h_{k-1}(u)$  are caused by two or more different elements  $u$ . Then the test indicates  $u \in S$  although this is not the case. Otherwise, if at least one bit position in the two Bloom filters varies,  $u$  clearly is no member of  $S$ .

## **2.1 PPRL with Bloom Filters Built from Multiple Identifiers**

In [15] Bloom filters were used in privacy-preserving record linkage for the first time. This approach was expanded to Cryptographic Longterm Keys in [16].

In common PPRL protocols two data owners A and B agree on a set of identifiers that occur in both of their databases. Next, these identifiers are standardized, then padded with blanks at the beginning and the end, and finally split into substrings of two characters. Each substring of the first identifier corresponding to a record is mapped to the first Bloom filter via several hash functions. Afterwards, each substring of the second identifier, corresponding to the same record, is mapped through another set of hash functions to the first Bloom filter as well. This procedure is repeated until all identifiers of the first record are stored in the first Bloom filter. Next, all identifiers corresponding to the second record of the database are mapped through the utilized hash functions to a second Bloom filter and so on. Performing this procedure for all entries of the database results in a set of Bloom filters where each Bloom filter is built from multiple identifiers. Thus, the similarity of the Bloom filters is a measure for the similarity of the encoded identifiers. Usually, the linkage of the two databases is conducted by a third party C.

Because of the specific structure of Bloom filters, record linkage based on Bloom filters built from multiple identifiers allows for errors in the encrypted data. Therefore, they can be applied to linking large data sets such as national medical databases [13].

## **2.2 Extant Research: Attacks on Bloom Filters of One or More Identifiers**

To the best of our knowledge, only two ways of attacking Bloom filters of one identifier and one way of attacking Bloom filters of multiple identifiers are known so far.

The first cryptanalysis of Bloom filters was published in 2011. Kuzu et al. [7] sampled 20,000 records from a voter registration list and encrypted the substrings of two characters from the forenames through 15 hash functions and Bloom filters of length 500 bits. Their attack consisted in solving a constraint satisfaction problem (CSP). Through a frequency analysis of the forenames and the Bloom filters and by applying their CSP solver to the problem, Kuzu et al. were able to decipher approximately 11% of the data.

In contrast, Niedermeyer et al. [11] proposed an attack on 10,000 Bloom filters built from encrypted German surnames that were considered to be a random sample of a known population. For the generation of the Bloom filters 15 hash functions and Bloom filter length 1,000 were used. Then they conducted a manual attack based on the frequencies of the substrings of length two, which they derived from the German surnames. Thus, Niedermeyer et al. deciphered the 934 most frequent surnames of 7,580 different ones, which corresponds to approximately 12% of the data set. However, their attack is not

limited to the most frequent names and could be extended to the decipherment of nearly all names.

In 2012 Kuzu et al. [8] showed an attack on Bloom filters built from multiple identifiers. They applied their constraint solver to forename and surname, as well as forename, surname, city and ZIP code, of 50,000 randomly selected records from the North Carolina voter registration list. However, they were not able to mount a successful attack. Thus, Kuzu et al. supposed that combining multiple personal identifiers into a single Bloom filter would offer a protection mechanism against frequency attacks. Although they suspected that their attack did not uncover all vulnerabilities of the Bloom filter encodings, they showed that the CSP for multiple identifiers is intractable to solve by their constraint solver.

### 2.3 Our Contribution

In this paper we present a fully automated attack on a database containing forenames, surnames and the relevant place of birth as well. All records are considered to be a random sample of a known population. We suppose that the attacker only knows some publicly available lists of the most common forenames, surnames and locations. The attack is based on analyzing the frequencies and the combined occurrence of substrings of length two from the identifiers of these lists. Furthermore, we are interested in recovering as many identifiers as possible. Our cryptanalysis was implemented using the programming languages Python and C++.

## 3 Encryption

In this section some basic notation is introduced and the encrypting procedure is described.

In record linkage scenarios, strings are usually standardized through transformations such as capitalization of characters or removal of diacritics [14]. After this *preprocessing step* all strings contain only tokens from some predefined alphabet  $\Sigma$ . Throughout this article, we use the canonical alphabet  $\Sigma := \{A, B, \dots, Z, \sqcup\}$ , where  $\sqcup$  denotes the padding blank. Thus, for example the popular German surname Müller is transformed to  $\sqcup\text{MUELLER}\sqcup$  in the preprocessing step. As usual, we denote substrings of two characters with *bigrams* and the set containing all the bigrams with  $\Sigma^2$ , i.e.

$$\Sigma^2 = \{\sqcup\sqcup, \sqcup A, \dots, \sqcup Z, A\sqcup, \dots, Z\sqcup, AA, \dots, ZZ\}.$$

The Bloom filter encryption of a record from a database is created by storing the bigram set associated with this record into a Bloom filter. The bigram set associated with a record is defined as the set containing the bigrams from all the identifiers. Here, a distinction between the bigrams occurring in different identifiers is made. Thus, if the set of identifiers is denoted with  $\mathcal{I}$ , the bigram set of a record is a subset of  $\mathcal{I} \times \Sigma^2$ .

For example, if we have  $\mathcal{I} = \{\text{surname}, \text{forename}\}$  and the database contains a record, Peter Müller, the bigram set  $\mathcal{S}_{\text{record}}$  associated with this record would contain the bigrams  $\sqcup P_f, P E_f, E T_f, T E_f, E R_f, R \sqcup_f, \sqcup M_s, M U_s, U E_s, E L_s, L L_s, L E_s, E R_s$  and  $R \sqcup_s$  (the subscript  $f$  indicates the bigrams occurring in the forename identifier, the subscript  $s$  the ones occurring in the surname identifier).

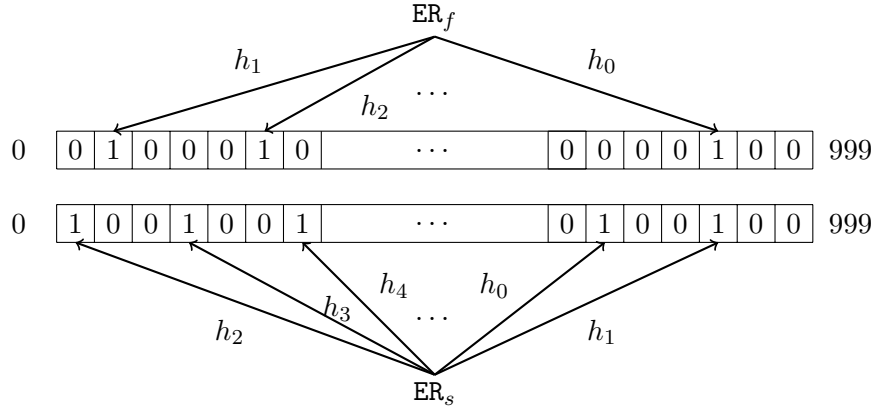


Figure 1: Two different atoms of the bigram **ER**. These atoms are realized when instances of **ER** occur in distinct identifiers.

Next, this bigram set is stored into a Bloom filter  $(b_0, \dots, b_{L-1})$  of length  $L$  by means of  $k$  independent hash functions

$$h_i : \mathcal{I} \times \Sigma^2 \rightarrow \{0, \dots, L-1\}$$

for  $i = 0, \dots, k-1$ . In practice, one could alternatively use different hash functions  $h_i : \Sigma^2 \rightarrow \{0, \dots, L-1\}$  for the distinct identifiers in order to guarantee that the hash values for distinct identifiers are not the same.

Further, as in [11] we introduce the term *atom* for the specific Bloom filters which occur as the fundamental building blocks of the encryption method.

**Definition 3.1** (Atom). Let  $L, k \in \mathbb{N}$  and some hash functions  $h_0, \dots, h_{k-1}$  be defined as above. Then, a Bloom filter

$$\mathcal{B} := (b_0, \dots, b_{L-1}) \in \{0, 1\}^L$$

is termed an *atom* if there exists a bigram  $\beta \in \mathcal{I} \times \Sigma^2$  such that  $b_j = 1 \Leftrightarrow h_i(\beta) = j$  for some  $i = 0, \dots, k-1$ . Such a Bloom filter is called the *atom realized by the bigram  $\beta$*  and denoted with  $\mathcal{B}(\beta)$ .

Thus, atoms are special Bloom filters. Since each bigram is hashed via each  $h_i$  for  $i = 0, \dots, k-1$ , at most  $k$  positions in an atom can be set to one.

By combining the atoms of the underlying bigram set of a record with the bitwise OR operation, the Bloom filter of a record is composed as

$$\mathcal{B}(\text{record}) = \bigvee_{\beta \in \mathcal{S}_{\text{record}}} \mathcal{B}(\beta),$$

where  $\bigvee$  denotes the bitwise OR operator.

Note that the same bigram from  $\Sigma^2$  is hashed differently if it occurs in distinct identifiers. This is illustrated in Figure 1 for the example of the bigram **ER** which occurs in the record **Peter Müller** both in the surname and the forename identifier.

Mapping each bigram of the forename **Peter** with  $k$  hash functions results in six atoms; for the surname **Müller**, we get eight atoms. Thus, the separate

	0000100000...0000000010	$\mathcal{B}(\sqcup P_f)$
∨	0001000001...0100000100	$\mathcal{B}(PE_f)$
∨	0101010101...0001010101	$\mathcal{B}(ET_f)$
∨	0001000010...0001000010	$\mathcal{B}(TE_f)$
∨	0100010001...0000000100	$\mathcal{B}(\textcolor{red}{ER}_f)$
∨	0101010101...0000000001	$\mathcal{B}(R_{\sqcup f})$
	<hr/>	
	0101110111...0101010111	$\mathcal{B}(\textbf{Peter})$
	0000000100...0000000001	$\mathcal{B}(\sqcup M_s)$
∨	0010000000...0100000000	$\mathcal{B}(MU_s)$
∨	0000100000...0010000010	$\mathcal{B}(UE_s)$
∨	1000000010...0010000000	$\mathcal{B}(EL_s)$
∨	0100001000...0100001000	$\mathcal{B}(LL_s)$
∨	1000000100...0001000000	$\mathcal{B}(LE_s)$
∨	1001001001...0000100100	$\mathcal{B}(\textcolor{red}{ER}_s)$
∨	0010001000...0000000010	$\mathcal{B}(R_{\sqcup s})$
	<hr/>	
	1111101111...0111101111	$\mathcal{B}(\textbf{Müller})$

Figure 2: Bloom filters of the forename **Peter** and the surname **Müller**, composed of the atoms belonging to the underlying bigrams.

	0101110111...0101010111	$\mathcal{B}(\textbf{Peter})$
∨	1111101111...0111101111	$\mathcal{B}(\textbf{Müller})$
	<hr/>	
	1111111111...0111111111	$\mathcal{B}(\textbf{entire record})$

Figure 3: The Bloom filter of the record **Peter Müller** is obtained by applying the bitwise OR operation to the Bloom filter encryptions of the separate identifiers.

Bloom filters for these identifiers might be composed as illustrated in Figure 2.

The final Bloom filter for the record **Peter Müller** is composed by applying the bitwise OR operation to the separate Bloom filter encryptions of the distinct identifiers. This is demonstrated in Figure 3.

In practice, the Bloom filter encryption of a record might contain a mixture of string valued identifiers (such as forename, surname or place of birth) and also numerical identifiers, such as date of birth. However, in this paper we restrict ourselves to the case of string valued attributes only, albeit our cryptanalysis proposed below is not limited to such attributes.

### Assumptions

In many record linkage scenarios, it is supposed that a semi-trusted third party conducts the record linkage between two encrypted databases. In this paper we assume a data set containing Bloom filters built from multiple identifiers that is sent to a semi-trusted third party. This third party acts as the adversary and tries to infer as much information as possible from the record encryptions.

We further suppose that the attacker has knowledge of the encryption process.

For our scenario we generated 100,000 Bloom filters built from standardized German forenames, surnames and cities according to the distribution in the population. The identifiers were truncated after the tenth letter, padded with blanks, respectively, and were broken into bigrams. Then the bigrams were hashed through  $k = 20$  hash functions into Bloom filters of length  $L = 1,000$ . As proposed in [15] and [16], we used the so-called *double hashing scheme* for the generation of  $k$  hash functions from two hash functions  $f$  and  $g$ . This double hashing scheme is defined via the equation

$$h_i = (f + i \cdot g) \bmod L \quad \text{for } i = 0, \dots, k-1 \quad (1)$$

and was originally proposed in [5] as a simple hashing method for Bloom filters yielding satisfactory performance results.

In our cryptanalysis we assume that the adversary knows that the hash values are generated in accordance with equation (1). It is self-evident that s/he must not have direct access to the hash functions  $f$  and  $g$  since this would permit the adversary to check whether a specific bigram is contained in a given Bloom filter.

Note that the double hashing scheme has also been used for the generation of Bloom filters by Kuzu et al. [8]. However, in that paper the knowledge of the *double hashing scheme* was not exploited in their cryptanalysis.

## 4 Cryptanalysis

This section provides a detailed description of the deciphering process. At first we try to detect the atoms that are contained in the given Bloom filters. Then, we assign bigrams to these atoms by means of an optimization algorithm. Finally, the original attributes are reconstructed from the atoms.

Our approach for the development of a fully automated attack is based on previous results on the automated cryptanalysis of simple substitution ciphers presented by Jakobsen [3]. We give a short account of Jakobsen's results in order to motivate our procedure.

### 4.1 Automated Cryptanalysis of Simple Substitution Ciphers

The encryption of a plaintext message through a *simple substitution cipher* is defined by a permutation of the underlying alphabet  $\Sigma$ . For instance, the message HELLO\_LISBON with tokens from the alphabet  $\Sigma = \{\_, \text{A}, \text{B}, \dots, \text{Z}\}$  could be encrypted as RVUUYJUOWAYL.

It is well known that this kind of encryption can be broken easily by means of a frequency analysis. However, just replacing the  $i$ -th frequent character in the ciphertext with the  $i$ -th frequent character in the underlying language will usually not lead to the correct decipherment (even for longer messages). This is commonly compensated for by taking bigram frequencies into consideration as well.

The expected bigram frequencies can be obtained from a training data set composed of the underlying language and stored in a quadratic matrix  $E$  (in the above example a  $27 \times 27$  matrix), where the entry  $e_{ij}$  is equal to the relative proportion of the bigram  $c_i c_j$  in the training text corpus and  $c_i$  denotes the



$i$ -th character of the alphabet. Analogously, the bigram frequencies of the ciphertext can be stored in a matrix  $D$ .

The algorithm proposed by Jakobsen [3] was intended to find a permutation  $\sigma_{\text{opt}}$  of the alphabet such that the objective function  $f$  defined via

$$f(\sigma) := \sum_{i,j} |d_{\sigma(i)\sigma(j)} - e_{ij}| \quad (2)$$

was minimized. The algorithm starts with the initial permutation that reflects the best assignment between single characters in the plaintext and the ciphertext with respect to their relative frequency. In each step of the algorithm two elements of the currently best permutation  $\sigma_{\text{opt}}$  are swapped, leading to a new candidate permutation  $\sigma$ . If  $f(\sigma) < f(\sigma_{\text{opt}})$  holds, the current permutation is updated to  $\sigma$ , otherwise  $\sigma$  is discarded and a new candidate  $\sigma$  is generated by swapping two other elements of  $\sigma_{\text{opt}}$ . This is repeated until no swap leads to a further improvement of the objective function  $f$ . Throughout this paper we use the same strategy as Jakobsen in [3], in order to determine the elements of the current permutation to be swapped. For a more detailed description of Jakobsen's method in the case of simple substitution ciphers we refer the reader to the original paper [3]. Figure 2 in [3] shows that a ciphertext of length 600 built by a simple substitution cipher can be entirely broken by this method. It is clear that some modification of Jakobsen's original algorithm is necessary in order to make it applicable in our setting as well. In particular, the definitions of the matrices  $D$  and  $E$  must be changed. Their adopted definitions are introduced in subsection 4.3.

## 4.2 Atom Detection

As in [11], the basic principle of our approach consists in the detection of atoms, which represent the encryption of one single bigram only. Since the Bloom filter of a string is created by the superposition of at least a few atoms, the reconstruction of the atoms given only a set of Bloom filters turns out to be difficult. Note that this task cannot be solved in a satisfactory manner if Bloom filters are considered isolatedly or in small groups because in this case too many binary vectors will be wrongly classified as atoms.

Let us give a short motivation for our novel method aiming at atom detection. If the bitwise AND operation is applied to a set of Bloom filters that have one bigram  $\beta$  in common, at least all positions set to one by  $\beta$  are equal to one in the result. However, for prevalent bigrams it should be expected that all the other positions are set to zero if a sufficient number of Bloom filters are considered, i.e., the result would be exactly the atom induced by the bigram  $\beta$ .

Of course, if an adversary has access to a set of Bloom filters, s/he does not a priori know which Bloom filters have a bigram in common. This obstacle can be avoided as follows: Under the assumption that the double hashing scheme is being used, the adversary is able to determine for each combination of bit positions from equation (1) the set of Bloom filters for which all these positions are set to 1. Then, the bitwise AND operation is applied to the set of these Bloom filters. If the result coincides with the atom, it is considered to be the realization of a bigram by the adversary.



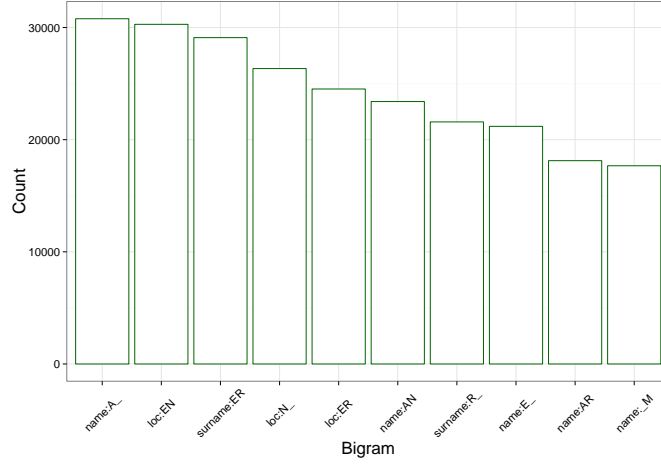


Figure 4: Absolute frequencies of the 10 most frequent bigrams in our training data set.

The resulting set of atoms was further reduced by discarding atoms of Hamming weight  $\sum_{i=0}^{999} b_i$  equal to 1, 2, 4 or 5 and keeping only atoms of Hamming weight equal to 8, 10 or 20.

Otherwise, too many binary vectors would have been classified incorrectly as atoms. The probability that an atom has Hamming weight less than 8 in our setting is equal to 0.008. This value can be derived in analogy to Lemma A.1 and the subsequent example in [11]. We denote the number of atoms found by  $n$ . For our specific data set we got  $n = 1,776$ . This result seems reasonable, because the total number of possible atoms is bounded from above by 2,187 and obviously not all of these atoms, in particular atoms realized by rare bigrams, occur in our simulated data. For each atom  $\alpha$  we determined the set of Bloom filters containing this atom, i.e. Bloom filters for which all bit positions of the atom are set to 1. We denote the atoms with  $\alpha_1, \dots, \alpha_{1776}$  according to decreasing frequency.

In the subsequent section we explain how correlations between the occurrences of atoms in the Bloom filters and bigrams in a training data set can be used to give adequate definitions of the matrices  $D$  and  $E$  that serve as the input of Jakobsen’s algorithm.

### 4.3 Correlation of Atoms and Bigrams

A naive assignment of bigrams to atoms is possible only for few frequent bigrams. For example, if German surnames, given names and birth locations are considered together, the most frequent bigram is  $A_{\square}f$  (the bigram  $A_{\square}$  in the forename identifier) such that the most frequent atom is likely to be the encryption of this bigram. The absolute frequencies of the 10 most frequent bigrams in the considered training data are illustrated in Figure 4.

Exept for the first few bigrams, the bigram frequencies are too close together such that naive matching is not promising for automatic decipherment. For this reason, we also took correlations between bigrams into account. For example, for records sampled from the population of Germany the appearance of the bigram  $CH_s$  in a record makes the appearance of the bigram  $SC_s$  in the same record more likely because the trigram  $SCH$  frequently appears in German surnames.

We model this kind of information on the correlation of atoms and bigrams by means of two matrices  $D$  and  $E$ . Assume that the attribution values of the records built from tokens of the alphabet  $\Sigma = \{\sqcup, A, B, \dots, Z\}$  are to be encrypted. Thus, for each (string valued) identifier we have 729 possible bigrams. Since the same bigram is encrypted differently for each identifier we have to distinguish between different instances of the same bigram. In our setting we denote the bigram  $\beta$  for the surname, forename and location identifier with  $\beta_s$ ,  $\beta_f$  and  $\beta_l$ , respectively. Altogether, the set  $\Sigma^2$  containing all possible bigrams consists of  $3 \cdot 729 = 2,187$  elements.

Let us now introduce the matrix  $E$  containing information about the expected bigram correlations obtained from the training data set. Note that the training data should be as similar to the encrypted data as possible, e.g. a random sample from the same underlying population as the encrypted data. If the prevailing Bloom filters are known to contain encryptions of records from the German population, an attacker would try to get access to a comparable database containing the same identifiers. The attribute values of this training data set are preprocessed analogously to the preprocessing routine before the encryption process. Then, the bigram sets for all the attribute values are created. We denote the bigrams with  $\beta_1, \dots, \beta_{2187}$  according to decreasing frequency. Let  $T$  be the total number of records in the training data set and  $t_{ij}$  the number of records that contain both bigram  $\beta_i$  and bigram  $\beta_j$ . Then the matrix  $E = (e_{ij})_{i,j=1,\dots,2187}$  is defined via

$$e_{ij} = \begin{cases} t_{ij}/T & \text{if } i \neq j, \\ 0 & \text{if } i = j. \end{cases}$$

The matrix  $D$  is formed in a similar way on the basis of joint appearances of atoms in the Bloom filters. Let  $N$  be the number of Bloom filters for which atoms have been extracted. We denote the number of Bloom filters that contain both atom  $\alpha_i$  and atom  $\alpha_j$  by  $b_{ij}$ . The matrix  $D = (d_{ij})_{i,j=1,\dots,2187}$  is defined through

$$d_{ij} = \begin{cases} b_{ij}/N & \text{if } i \neq j \text{ and } i, j \leq 1776, \\ 0 & \text{if } i = j \text{ or } \max(i, j) > 1776. \end{cases}$$

The procedure suggested by Jakobsen which was described above can now directly be applied to the matrices  $D$  and  $E$ . The pseudocode for the overall algorithm can be found in algorithm 1.

The progress of the optimization algorithm is illustrated by means of Figure 5.

The result of the algorithm will be the final assignment between atoms and bigrams defined by a permutation  $\sigma_{\text{opt}} \in S_{2187}$  and the assignment rule  $\alpha_{\sigma_{\text{opt}}(i)} \rightarrow \beta_i$ . This assignment is used to reconstruct the original bigram sets encrypted in the Bloom filters.

For example, the bigrams  $\text{HE}_l$ ,  $\text{E}_{\sqcup l}$ ,  $\sqcup K_l$ ,  $\text{RL}_l$ ,  $\text{AR}_l$ ,  $\text{LS}_l$ ,  $\text{RU}_l$ ,  $\text{KA}_l$ ,  $\text{UH}_l$ ,  $\text{SR}_l$ ,  $\text{ER}_s$ ,  $\text{R}_{\sqcup s}$ ,  $\text{CH}_s$ ,  $\text{SC}_s$ ,  $\text{HE}_s$ ,  $\sqcup F_s$ ,  $\text{IS}_s$ ,  $\text{ID}_s$ ,  $\text{FI}_s$ ,  $\text{N}_{\sqcup f}$ ,  $\sqcup S_f$ ,  $\text{ON}_f$ ,  $\text{SI}_f$ ,  $\text{IM}_f$  and  $\text{MO}_f$  were assigned to the Bloom filter No. 850.

In the following section we describe how attribute values are reassembled from the reconstructed bigram sets.

---

**Algorithm 1** OPTIMIZATION ALGORITHM

---

**Input:**  $D, E$  as defined in section 4.3

**Output:**  $\sigma_{\text{opt}} \in S_{2187}$  minimizing

$$f(\sigma) = \sum_{i,j} |d_{\sigma(i)\sigma(j)} - e_i e_j|$$

```

1:  $\sigma_{\text{opt}}(i) = i \ \forall i$  ▷ Initialization
2:  $\min \leftarrow f(\sigma_{\text{opt}})$ 
3:  $a, b \leftarrow 1$ 
4: repeat
5:    $\sigma \leftarrow \sigma_{\text{opt}}$ 
6:    $a \leftarrow a + 1$ 
7:   if  $a + b \leq 2187$  then
8:      $\sigma(a) \leftarrow \sigma_{\text{opt}}(b), \sigma(b) \leftarrow \sigma_{\text{opt}}(a)$ 
9:   else
10:     $a \leftarrow 1, b \leftarrow b + 1$ 
11:   if  $f(\sigma) < f(\sigma_{\text{opt}})$  then ▷ Update
12:      $\min \leftarrow f(\sigma)$ 
13:      $\sigma_{\text{opt}} \leftarrow \sigma$ 
14:      $a, b \leftarrow 1$ 
15: until  $b = 2187$ 

```

---

#### 4.4 Reconstruction of Attribute Values

In order to reconstruct the original attribute values of the records, we separated the bigrams belonging to different identifiers for each Bloom filter. Then, our approach to reconstructing the original identifier values was to compare the obtained bigram sets with a list of bigram sets generated from reference lists of surnames, names and locations. For Bloom filter No. 850, for example, an adversary would correctly guess that this Bloom filter encrypts a record belonging to the person **Simon Fischer** from the German city **Karlsruhe**.

#### 4.5 Results

By using the approach described above, we were able to reconstruct 59.6% of the forenames, 73.9% of the surnames and 99.7% of the locations correctly. For 44% of the 100,000 records all the identifier values were recuperated successfully.

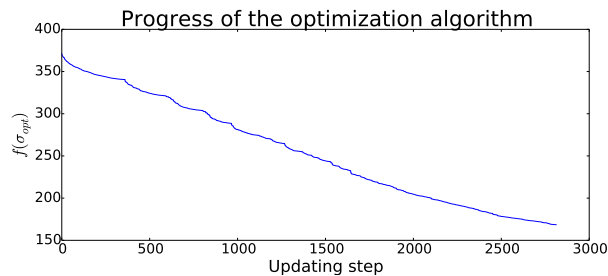


Figure 5: Progress of the optimization algorithm for our data set. The initial value of the objective function is 370.99 and 2,812 updating steps were performed. The final value of the objective function  $f(\sigma_{\text{opt}})$  was equal to 168.5.

## 5 Conclusion

In this paper we demonstrate a successful fully automated attack on Bloom filters built from multiple identifiers. We were able to recover approximately 77.7 % of the original identifier values. In contrast to the assumptions in [8] and [11], that storing all identifiers in a single Bloom filter makes it more difficult to attack, we needed only moderate computational effort and publicly available lists of forenames, surnames, and locations to reconstruct the identifiers. Note that there is no huge impact of the size of the database containing the Bloom filters. For our cryptanalysis it is sufficient to perform the attack on a subset of the given Bloom filters (100,000 as in our example should be adequate in most cases). Then for the remaining Bloom filters it would be sufficient to check for the atoms contained in those and to reconstruct the attribute values, since most assignments of atoms to bigrams are already known. Thus, the time needed for cryptanalysis is linear in the number of input Bloom filters. The time needed for the detection of atoms is  $O(L^2)$  since there are  $L$  possible values for the hash functions  $f$  and  $g$  in equation (1). Furthermore, the detection of atoms could easily be parallelized to make the computation faster and values of  $L$  significantly larger than  $L = 1,000$  as considered in this paper would also have negative effects on the time needed for performing the linkage between two databases (note that in the large scale study reported in [13] a Bloom filter length of only 100 was considered). Thus, the most time consuming step in our cryptanalysis should be the optimization algorithm presented in subsection 4.3. Indeed, in the chosen parameter setup this procedure took about 402 minutes on a notebook with 2.80 GHz Intel<sup>®</sup> Core running Ubuntu 14.04 LTS.

To sum up, we do not recommend the usage of Bloom filters built from one or more identifiers, generated with the double hashing scheme, in applications where high security standards are required. However, we applied our attack in a very special scenario, because the generated databases were encrypted using the double hashing scheme. Thus, there are options for an improvement of the setting.

For example Niedermeyer et al. [11] proposed several methods such as fake injections, salting or randomly selected hash values to harden the Bloom filters. Hence, we are confident that methods like those proposed by Niedermeyer et al. show promise in the prevention of attacks like the one presented in this paper.

## Acknowledgements

This research has been partly supported by the grant SCHN 586/17-1 of the German Research Foundation (DFG) awarded to Rainer Schnell.

## References

- [1] Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426.
- [2] Herzog, T. N., Scheuren, F. J., and Winkler, W. E. (2007). *Data Quality and Record Linkage Techniques*. Springer, New York.
- [3] Jakobsen, T. (1995). A fast method for the cryptanalysis of substitution ciphers. *Cryptologia*, 19(3):265–274.
- [4] Jones, M., McEwan, P., Morgan, C. L., Peters, J. R., Goodfellow, J., and Currie, C. J. (2005). Evaluation of the pattern of treatment, level of anticoagulation control, and outcome of treatment with warfarin in patients with non-valvar atrial fibrillation: a record linkage study in a large British population. *Heart*, 91(4):472–477.
- [5] Kirsch, A. and Mitzenmacher, M. (2008). Less hashing, same performance: Building a better Bloom filter. *Random Structures & Algorithms*, 33(2):187–218.
- [6] Kuehni, C. E., Rueegg, C. S., Michel, G., Rebholz, C. E., Strippoli, M.-P. F., Niggli, F. K., Egger, M., and von der Weid, N. X. (2012). Cohort profile: The Swiss childhood cancer survivor study. *International Journal of Epidemiology*, 41(6):1553–1564.
- [7] Kuzu, M., Kantarcioglu, M., Durham, E., and Malin, B. (2011). A constraint satisfaction cryptanalysis of bloom filters in private record linkage. In Fischer-Hübner, S. and Hopper, N., editors, *Privacy Enhancing Technologies*, volume 6794 of *Lecture Notes in Computer Science*, pages 226–245. Springer, Berlin.
- [8] Kuzu, M., Kantarcioglu, M., Durham, E. A., Toth, C., and Malin, B. (2012). A practical approach to achieve private medical record linkage in light of public resources. *Journal of the American Medical Informatics Association*, 20(2):285–292.
- [9] Napoleão Rocha, M. C. (2013). *Vigilância dos óbitos Registrados com Causa Básica Hanseníase*. Master thesis, Universidade de Brasília, Brasília.
- [10] Newman, T. B. and Brown, A. N. (1997). Use of commercial record linkage software and vital statistics to identify patient deaths. *Journal of the American Medical Informatics Association*, 4(3):233–237.
- [11] Niedermeyer, F., Steinmetzer, S., Kroll, M., and Schnell, R. (2014). Cryptanalysis of basic Bloom filters used for privacy preserving record linkage. Working Paper NO.WP-GRIC-2014-04, German Record Linkage Center, Nürnberg.
- [12] Office for National Statistics (2013). Beyond 2011: Matching anonymous data. Methods & Policies M9, ONS, London.
- [13] Randall, S. M., Ferrante, A. M., Boyd, J. H., Bauer, J. K., and Semmens, J. B. (2014). Privacy-preserving record linkage on large real world datasets. *Journal of Biomedical Informatics*.
- [14] Randall, S. M., Ferrante, A. M., Boyd, J. H., and Semmens, J. B. (2013). The effect of data cleaning on record linkage quality. *BMC Medical Informatics and Decision Making*, 13(64).
- [15] Schnell, R., Bachteler, T., and Reiher, J. (2009). Privacy-preserving record linkage using Bloom filters. *BMC Medical Informatics and Decision Making*, 9(41):1–11.

- [16] Schnell, R., Bachteler, T., and Reiher, J. (2011). A novel error-tolerant anonymous linking code. Working Paper NO.WP-GRLC-2011-02, German Record Linkage Center, Nürnberg.
- [17] Schnell, R., Richter, A., and Borgs, C. (2014). Performance of different methods for privacy preserving record linkage with large scale medical data sets. Presentation at International Health Data Linkage Conference, Vancouver.
- [18] Van Den Brandt, P. A., Schouten, L. J., Goldbohm, R. A., Dorant, E., and Hunen, P. M. H. (1990). Development of a record linkage protocol for use in the Dutch cancer registry for epidemiological research. *International Journal of Epidemiology*, 19(3):553–558.